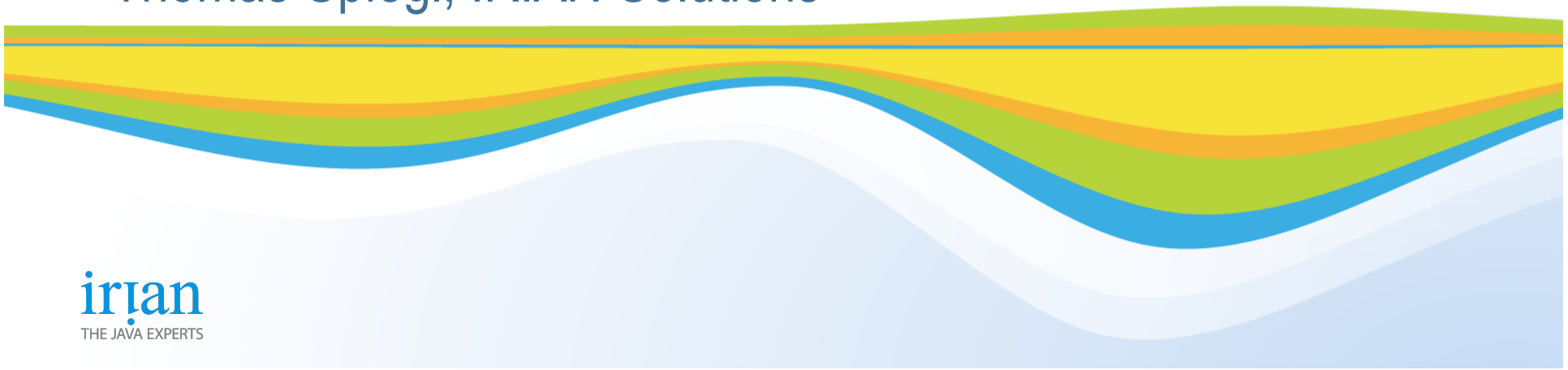
# JSF 2

## The good, the bad and the ugly

Thomas Spiegl, IRIAN Solutions



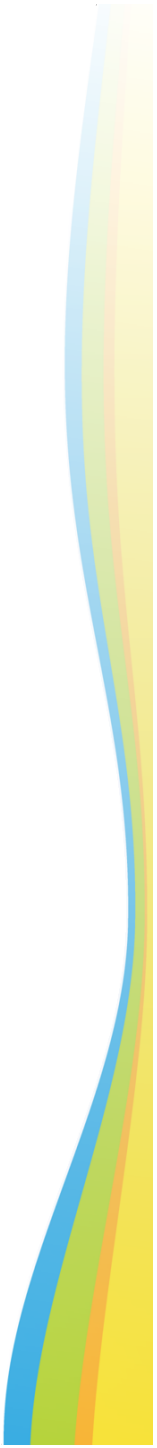
irian
THE JAVA EXPERTS

# Thomas Spiegl

- CEO IRIAN Solutions GmbH
- Gründer Apache MyFaces 2002
- Committer Apache MyFaces
- Architektur und Design von JEE Applikationen
- Vorträge und Schulungen zum Thema JSF

Die Qual der Wahl

# KOMPONENTEN



# JSF Komponenten

- Welche Bibliothek soll verwendet werden?
- Lassen sich 2 oder mehr Bibliotheken gemeinsam einsetzen?
- Gibt es langfristigen Support für die Bibliothek?

# ☹ Best you can do (ironic)

- Möglichst viele verschiedene Komponenten-Bibliotheken mischen
  - IceFaces + Richfaces + MyFaces Trinidad – excellent!
  - Skinning 3x implementieren
  - Kompatiblitätsprobleme garantieren meinen Job, oder ich darf bald die Stelle wechseln
  - Ein Release-Upgrade und haben wir haben alle Hände voll zu tun

Request oder Session?

# SCOPE

# ☹ SessionScoped

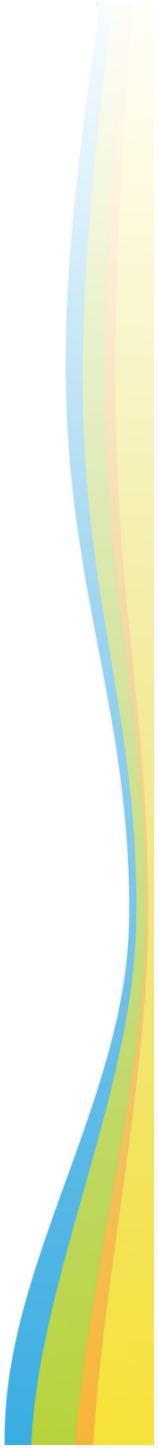
- Annahme
  - Speicher ist ausreichend vorhanden, 10 MB pro UserSession sind total ok!
  - Multi-Window Support ist nicht von Interesse!
  - Sollte ein Benutzer die Applikation trotzdem mit mehr als einem Fenster bedienen wollen, dann auf eigene Gefahr!
- Gut, packen wir alles in die HttpSession ☺

# ☺ ConversationScoped

- Am Ende der Conversation wird die HttpSession bereinigt
- Multi Window Support
- Lösungen
  - JSR 299 (OpenWebbeans, Weld)
  - MyFaces Orchestra + Spring
  - Spring Webflow

Mal so oder auch anders

# NAMING CONVENTIONS



# ☹ Naming Convention

- Keine Namenskonvention für XHTML-Seiten und Managed Beans
- Wir wollen die Kreativität des Einzelnen nicht untergraben

```
person.xhtml : PersonDetailBacking.java

kunde.xhtml  : ClientBean.java
```

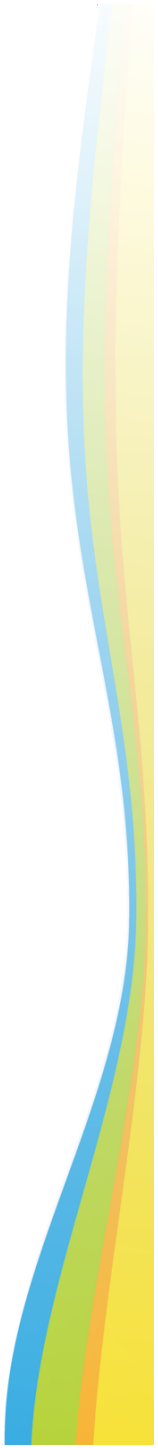
# ☺ Naming Convention

- Name der XHTML-Seite lässt auf das entsprechende Backing-Bean schliessen

```
personList.xhtml : PersonListPage.java

personDetail.xhtml : PersonDetailPage.java
```

Gut zu wissen.

# FACES CONTEXT



# ☹ Cache FacesContext

```
@javax.faces.bean.ManagedBean
@javax.faces.bean.SessionScoped
public class PageBean {

  private FacesContext context;

  public FacesContext getFacesContext() {
    if (context == null) {
        context = FacesContext.getCurrentInstance();
    }
  }
}
```

Request Objekt
im SessionScope!

irian
THE JAVA EXPERTS

# ☹ Reference Beans

```
@javax.faces.bean.ManagedBean
@javax.faces.bean.SessionScoped
public class PageBean {

  public String getSomeValue() {
    return
      JSFUtils.getBean("otherPageBean").getSomeValue();
  }
}
```

☹ Wartung
☹ Performance

# ☺ Reference Beans

```
@javax.faces.bean.ManagedBean
@javax.faces.bean.SessionScoped
public class PageBean {

  @ManagedProperty(value="otherBean")
  private OtherBean otherBean;

}
```

☺ IDE Support
☺ Performance

Mit Version 2 in JSF endlich angekommen.

# AJAX

# ☹ JSF + Ajax

- Ajax ist so cool, verwenden sooft es geht

```
<h:inputText value="">
  <f:ajax event="onkeypress" render="output" />
</h:inputText>
<h:outputText id=”output” value="" />
```

- Seitennavigationen mit Ajax
  - Browser Back
  - Page Refresh

# ☺ Ajax

- Sinn- bzw. massvoll einsetzen
  - Tabelle filtern
  - Dependend Dropbox
  - Autocompletion

Von A nach B und wieder retour

# NAVIGATION

# ☹ Navigation

- Kein Redirect nach Post
  - Browser Back löst einen Post Request
  - ViewExpiredEcxeption führen
  - Nach Aufruf einer Action Methode führt ein Browser-Refresh oder Browser-Back zum erneuten Aufruf dieser

# ☹ Navigation

- Unklare bzw. fehlende Namenskonvention für Navigation Outcomes
  - “back”, “retour”, “saved”, ”persons”

# ☺ Navigation

- Verwendung des Redirect-Post Patterns
- Redirect für jeden Navigationcase
  - Erneuter Aufruf der Action wird vermieden
  - URL entspricht der angezeigten Seite
- Zielseite direkt angeben

Wozu in XML deklarieren?

```
<h:commandButton value="Cancel"
   action="/cancelled.xhtml?faces-redirect=true"/>
```

Ein wichtiges Thema!

# EXCEPTION HANDLING

# ☹ Exception Handling

```
@ManagedBean @SessionScoped
public class PersonDetailPage {
  @EJB
  private PersonServiceFacade personServiceFacade;
  private Person person;

  public String save() {
    try {
      personServiceFacade.savePerson(person);
    } catch (DomainException e) {
       ....
    }
    return null;
  }
```

# ☹ Exception Handling

- Copy & Paste
- Uneinheitliches Exception Handling
- Wartung!!!

# ☺ Exception Handling

- Use unchecked Exceptions

```
public class DomainExeption extends RuntimeExcption {…}
```

# ☺ Central Exception Handling

```
@Aspect
public class ExceptionMethodInterceptor {

  @Around("execution(public * at.irian.web.*.*(..))")
  public Object intercept(ProceedingJoinPoint pjp) throws
Throwable {
    try {
      return pjp.proceed();
    } catch (DomainException e) {
       FacesMessage msg = new FacesMessage(
         FacesMessage.SEVERITY_ERROR, "Fehler" e.getMessage());
       FacesContext.getCurrentInstance()
         .addMessage(null, msg);
    }
```

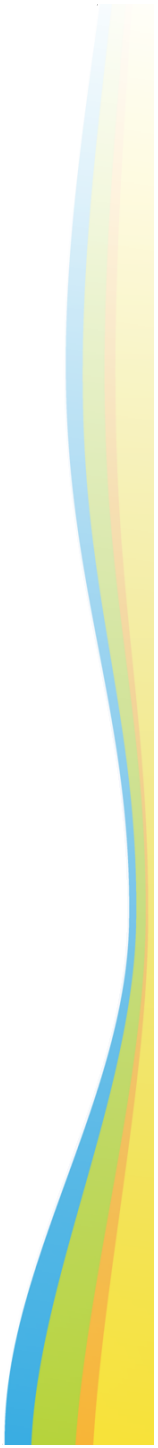
# ☺ Central Exception Handling

```
package at.irian.web.person;
public class PersonDetailPage {

  @EJB
  private PersonServiceFacade personServiceFacade;
  private Person person;

  public String save() {
    // PersonServiceFacade may throw a DomainException
    // which will be handled by ExceptionMethodInterceptor
    personServiceFacade.savePerson(person);
    return null;
  }
}
```

irian
THE JAVA EXPERTS

Es geht zu langsam!

# PERFORMANCE



# ☹ Heavy use of expressions in Tables

```
<h:column>
 <h:outputText rendered="#{page.cond1 or row.type eq 't' }"/>
 <h:outputText rendered="#{page.cond1 or row.value > 10}"/>
</h:column>
```

- Schwer lesbar
- Rendered Expression wird für jede Zeile mehrfach ausgewertet (mehrere Lifcycle Phasen!)

# ☹ Client Side State Saving

```
<context-param>
  <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
  <param-value>client</param-value>
</context-param>
```

- Unsafe as long as not decrypted
- Consumes quite some bytes for every request/ response (also Ajax requests!)

# ☺ Server Side State Saving

- Better performance

```
<context-param>
  <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
  <param-value>client</param-value>
</context-param>
<context-param>
  <param-name>
      org.apache.myfaces.SERIALIZE_STATE_IN_SESSION
  </param-name>
  <param-value>false</param-value>
</context-param>
```

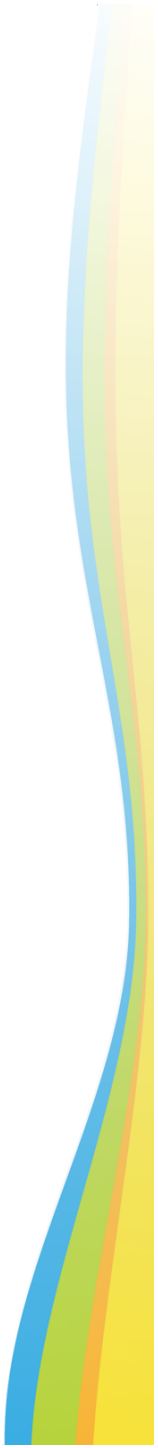
# Performance Probleme

- Es gibt unzählige Problemfälle
- Geht's zu langsam, so muss ein Profiler her
- Lässt exakt auf das Problem schliessen

- Tool 

Diverse

# EMPFEHLUNGEN



# ☺ Tips

- <h:messages /> im Template
  - JSF 2.0 fügt Messages im Development automatisch hinzu

- Implement equals und hashcode Methoden für SelectItems

# ☹ faces-config.xml

```
<context-param>
  <param-name>javax.faces.CONFIG_FILES</param-name>
  <param-value>/WEB-INF/faces-config.xml</param-value>
</context-param>
```

- Faces Config wird 2x eingelesen
- Vorhandener NavigationListener wird 2x registriert

# ☹ Bind Rendered to RequestScope

```
<h:commandButton action="#{page.save} value="Save"
   rendered="#{page.renderSaveButton}" />
```

- True during rendering
- False on postback to server
- Action will not be triggered!

# ☹ Ableitung aus Faulheit

```
@ManagedBean
public class PersonListPage extends AbstractListPage<Person> {

    @Overrides
    public List<PersonVO> executeQuery() {...}

    @Overrides
    public Filter<PersonFilter> createFilterInstance() {...}
}
```

- Funktion ist auf eine bestimmte Seite gebunden
- Wiederverwendung nur schwer möglich

# ☺ Delegate statt Extend

```
@ManagedBean
public class ComplexPage {
   // delegate to specialized classes
   private PersonListSupport personListSupport;
   private AddressListSupport adressListSupport;
}

public class PersonListSupport extends ListSupport<Person> {
    @Overrides
    public List<PersonVO> executeQuery () {...}

    @Overrides
    public Filter<PersonFilter> createFilterInstance () {...}
}
```

# ☺ JSF Lifecycle

- Noch nie gehört
- Vielleicht schau ich mir das später mal an

# ☺ JSF Lifecycle

- Der JSF Lifecycle ist das Herzstück von JSF und sollte von jedem JSF Entwickler verstanden werden
- Viele Probleme lassen sich leichter lösen, wenn der Lifecycle verstanden wurde

# ☹ IDE

- Don’t use and IDE that supports code completion and shows errors in JSF pages
- Never spend money on a good IDE, Eclipse is for free
- Never refactor your code, as it’s too much work to do everything by hand
- Last but not least: Don’t use a Debugger

2 Standards im Einsatz

# JPA + JSF

# JPA - Entity

```
@Entity
public class DebitItem {
  private Integer number;

  @ManyToOne(fetch = FetchType.EAGER)
  private Debtor debtor;

  @ManyToOne(fetch = FetchType.LAZY)
  private Creditor creditor;

}
```

# ☹ Use Entity in DataTable

```
<h:dataTable value="#{page.debitItems} var="item">
  <h:column>
    <h:outputText value="#{item.number}"
  </h:comlumn>
  <h:column>
    <h:outputText value="#{item.debitor.name}"
  </h:comlumn>
  <h:column>
    <h:outputText value="#{item.creditor.name}"
  </h:comlumn>
</h:dataTable>
```

# ☺ Use Entity in DataTable

- Auch Daten die nicht benötigt werden, werden gelesen
- Bei FetchType.LAZY werden Daten pro Zeile nachgelesen
- Wenn das Entity geändert wird, hat dies unmittelbaren Einfluss auf das UI (Refactoring, Wartung wird erschwert)
- FetchType.LAZY kann für Detachtes Entity zu Problemen führen (Hibernate)

# ☺ Value Object

```
public class DebitItemVO {

  private String itemNumber;
  private String debtorName;
  private String creditorName;

  public DebitItemVO(String itemNumber, String debtorName,
                     String creditorName) { ... }

  // getter
  // no setter as it's a Value Object
}
```

# ☺ JPA Projection

```
public class DebitItemRepository {

  @PersistenceContext
  private EntityManager em;

  public List<DebitItemVO> getDebitItems() {
    em.createQuery(
      "select " +
      "new DebitItemVO(i.number, i.debtor.name, i.creditor.name)
" +
      "from DebitItem i").getResultList();
  }
}
```

# ☺ Use Value Object

```
<h:dataTable value="#{page.debitItems} var="item">
  <h:column>
    <h:outputText value="#{item.itemNumber}"
  </h:comlumn>
  <h:column>
    <h:outputText value="#{item.debtorName}"
  </h:comlumn>
  <h:column>
    <h:outputText value="#{item.creditorName}"
  </h:comlumn>
</h:dataTable>
```

# ☺ Use Value Objects

- Bessere Performance
- Wartungs- und Refactoring freundlich (Entity kann geändert werden)

# ☹ What's going wrong here?

```
@Entity
public class Client {

  @OneToMany(fetch = FetchType.EAGER)
  private Set<PriceReductions> conditions;

  @OneToMany(fetch = FetchType.EAGER)
  private List<Limits> limits;

}

em.createQuery("select c from Client c where id=1");
```

# Answer

- In der Datenbank
  - Client mit der Id 1 hat 10 PriceReduction- und 5 Limite-Einträge
  - Resultset hat 10 * 5 = 50 Zeilen
  - Bad Performance!!!
- Lösung 1

```
@OneToMany(fetch = FetchType.EAGER)
@Fetch(FetchMode.SELET)
private Set<PriceReductions> conditions;
```

- Lösung 2
  - RootEntities für PriceReductions und Limits

Es gäbe noch viel zu sagen

# TIME IS GETTING SHORT

# Architektur

- Architektur muss einfach sein
- Achtung: Annotation jungle – don't get lost
  - Annotationen im Projekt per Konvention stark einschränken!
- Keine JPA Entities im UI
  - Entities in Facade auf DTO und VO mappen

    Am einfachsten per Hand - Finger weg von Object Clonern!
  - Refactoring der Domainlogik wird enorm erleichtert
  - Ausnahme: Reine Datenschleudern (ohne Logik)


The annual **Con**ference **F**or **E**nterprise **S**oftware **S**olutions

We are looking forward to meeting you in Vienna - 12-14 April
http://www.con-fess.com